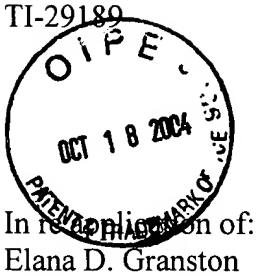
TI-29189



BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re application of:
Elana D. Granston

Serial No. 09/733,254

Title: Method for Software pipelining of Irregular Conditional Control Loops
Filed: December 8,2000

Examiner: VU, Tuan A.
Art Unit: 2124

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

MAILING CERTIFICATE UNDER 37 C.F.R. §1.8(A)
I hereby certify that the above correspondence is being deposited with the U.S. Postal Service as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Robert L. Troike 10/12/04
Robert L. Troike, Reg. No. 24,183 Date

Dear sir:

APPEAL BRIEF

As required by 37 C.F.R. 192 (c), this brief, filed in triplicate, contains the following items under appropriate headings and in the order there indicated. Please charge the cost of this brief and any other costs associated with this brief to deposit account no.20-0668 of Texas Instruments Inc. An oral hearing is not requested.

REAL PARTY IN INTEREST

The party of interest is Texas Instruments Incorporated of Dallas Texas, a Delaware corporation.

RELATED APPEALS AND INTERFERENCES

There are no other related appeals or interferences.

STATUS OF CLAIMS

Claim 1 is rejected under 35 U.S. C.112, first and second paragraph.

Claims 1-4 are rejected under 35 U.S.C. 103 (a) as being unpatentable over Rau et al.,

"Code Generation Schema for Modulo Scheduled Loops" , ACM Proceedings of the 25th annual International Symposium on Microarchitecture, Dec. 1992, hereinafter Rau_1) in view of Rau et al., " Register Allocation for Software Pipelined Loops", June 1992, In Proc. Of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2) and further in view of Akkary (US Patent no.6,240,509 (hereinafter Akkary) and Bringmann, " Enhancing Instruction Level Parallelism through Computer-Controlled Speculation", University of Illinois, 1995 (hereinafter Bringmann).

STATUS OF AMENDMENTS

An amendment after final was filed on July 9, 2004. Claim 1, was amended, to remove the objected to language in the 112 rejection of without special hardware support or special loop control instruction. It was not entered.

SUMMARY OF INVENTION

This invention relates to computers and computer program compliers and more particularly to pipelining program loops have irregular loop control. A loop consists of multiple iterations. Normally, the first iteration is completed before the second iteration, etc. The software pipelining optimization exploits the parallelism in a parallel architecture by initiating the next loop iteration before the first completes and so forth. However, care must be taken that we don't execute more iterations than existed in the original loop. With regular "for loops", we can cut off the initiation of future iterations at the right time because we know when and how close we are to being done. With irregular loop such as "while" loops there is no early warning that the end is near. We don't know we're on the last iteration until the condition changes. Hence, we may have already

initiated additional iterations that are past the end of the original loop. As mentioned in the background it has been known how to use special-purpose hardware to support pipelining for WHILE loops. See applicant's specification reference on page 4, lines 15-23 and referenced article of Tirumalai et al. For many applications this hardware is expensive in terms of cost or power or simply not available. It is also known how to handle the case where the compiler has insufficient knowledge to guarantee safety criteria for software pipelined loop. This problem is handled at compile time by generating two versions of the loop and using run-time trip count check to choose between them. This multi-version code generation increases code size and adds run time overhead. See applicant's specification reference on page 4 beginning on line 29 through page 5, lines 1-12 and referenced article of Monica Lam.

Applicant's proposed solution is to preprocess the original loop so that the additional iterations can be safely speculatively executed, which means that we can initiate extra iterations without the danger of generating incorrect program results. In particular, if it is safe to execute an instruction extra times, we leave it alone. If not, register copying is used to transform the instruction into one that can be speculatively executed. Applicant's claim 1 calls for a method of pipelining program loops having irregular loop control that comprises determining which instructions in loop code in a memory may be speculatively executed without special hardware support or special loop control instruction (Step N7 in Fig. 5), storing in a computer memory a set of registers that are modified by an instruction and are alive out of the loop (Steps N7, N9 and N11) and in Fig. 5 Step S1), and modifying the program code so that the values of those registers are saved to a temporary register during all proper iterations (in Fig 5, Step S3),

and copying back to a register the value of the temporary register once the loop is completed (Step S4). In other words, the results are copied to a temporary register or registers after it has been determined that the instruction would have been executed in the original stream. After loop execution, the original resister(s) are restored with the last value(s) written in the temporary register. Register copying is preferred over predication on architectures without special hardware support because the resulting WHILE loop can be pipelined more efficiently. Because there are typically very few live out registers in loops, register copying can generally be profitably applied, even on architecture with small register sets. After pre-processing a WHILE loop, the loop can be software pipelined similar to a traditional loop except that no pipe-down stage is needed, so there is no epilog. The minimum trip count to safely execute the loop is always one. There is no need for multi-version loops. According to Claim 2 it is the method of Claim 1, applied to any program loops wherein a minimum trip count is reduced to one. According to Claim 3 it is the method of Claim 1, applied to any program loops wherein a need for a multi-version code is eliminated.

Claim 4 is an independent claim that describes a method for software pipelining of irregular conditional control loops includes pre-processing the loops so they can be safely software pipelined, comprising pre-processing each instruction in the loop in turn. If the instruction can be safely speculatively executed, leaving the instruction alone. If it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction using register copying and otherwise using predication. Predication is the process of applying a guard to the instruction so that it is conditionally nullified. If it was already predicated, we compute a new predicate which is

the end of the old predicate and the predicate that controls looping and use that predicate instead.

ISSUES

ISSUE 1

Claim 1 is rejected under 35 U.S.C. §112, first paragraph, as failing to comply with the written description requirement.

ISSUE 2

Claim 1 is rejected under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly pointing out and distinctly claiming the subject matter which applicant regards as the invention.

ISSUE 3

Claims 1-4 are rejected under 35 U.S.C. 103 (a) as being unpatentable over Rau et al., "Code Generation Schema for Modulo Scheduled Loops" , ACM Proceedings of the 25th annual International Symposium on Microarchitecture, Dec. 1992, hereinafter Rau_1) in view of Rau et al., " Register Allocation for Software Pipelined Loops", June 1992, In Proc. Of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2) and further in view of Akkary (US Patent no.6,240,509 (hereinafter Akkary) and Bringmann, " Enhancing Instruction Level Parallelism through Computer-Controlled Speculation", University of Illinois, 1995 (hereinafter Bringmann).

GROUPING OF CLAIMS

These claims do not stand or fall together for the reasons presented in the Argument.

ARGUMENT

ISSUE 1

Claims 1 is rejected under 35 U.S.C.§112, first paragraph, as failing to comply with the written description requirement. The applicant has the language of "without special hardware support or special loop control instruction" on line 4. The examiner argues that this is not in the specification. Nowhere in applicant's specification is there any mention of special hardware support or a special loop instruction. This is not so. See applicant's specification reference on page 4, lines 15-23 and referenced article of Tirumalai et al. For many applications this hardware is expensive in terms of cost or power or simply not available. Also see the reference to generating two versions of the loop and using run-time trip count check to choose between them. This multi-version code generation increases code size and adds run time overhead. See applicant's specification reference on page 4 beginning on line 29 through page 5, lines 1-12 and referenced article of Monica Lam. Also see reference to this on page 9, lines 25-29.

This was put in to emphasize the difference from that in the prior art which has such special hardware support and loop instructions. Since the applicants did not rely solely on this distinction, applicant attempted to remove this by the amendment after final to reduce the issues on appeal but it was not entered.

ISSUE 2

Claim 1 is rejected under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly pointing out and distinctly claiming the subject matter which applicant regards as the invention. This again relates to the language of "without special hardware support or special loop control instruction" on line 4. This was put in to emphasize the difference from that in the prior art which has such special hardware support and loop instructions. See applicant's specification reference on page 4, lines 15-23 and referenced article of Tirumalai et al. For many applications this hardware is expensive in terms of cost or power or simply not available. Also see the reference to generating two versions of the loop and using run-time trip count check to choose between them. This multi-version code generation increases code size and adds run time overhead. See applicant's specification reference on page 4 beginning on line 29 through page 5, lines 1-12 and referenced article of Monica Lam. Also see reference to this on page 9, lines 25-29.

Since the applicants did not rely solely on this distinction, applicant attempted to remove this by the amendment after final to reduce the issues on appeal but it was not entered.

ISSUE 3

Claims 1-4 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rau et al " Code Generation Schema for Modulo Scheduled Loops", ACM Proceedings of the 25th annual International Symposium on Microarchitecture, Dec. 1992, volume 23,

hereinafter Rau_1 in view of Rau et al., "Register Allocation for Software Pipelines Loops", June 1992, In Proc. of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2), and further in view of Akkary, USPN 6,240,509 (hereinafter Akkary) and Bringmann, "Enhancing Instruction Level Parallelism through compiler-controlled speculation", Univ. of Illinois, 1995 (hereinafter Bringmann).

Applicant's claim 1 call for: "A method of software pipelining program loops having irregular loop control comprises the steps of:

determining which instructions in loop code in a memory may be speculatively executed,

storing in a computer memory a set of registers that are modified by an instruction and are alive out of the loop, and

modifying the program code so that the values of those registers are saved to a temporary register during all proper iterations, and

copying back to a register the value of the temporary register once the loop is completed."

This method of preprocessing as claimed above is not taught by the combination of references.

In Rau_1, the authors do claim that an irregular loop can be software pipelined if the entire epilog can be collapsed. While applicant's achieve this, this is not applicant's invention.

The primary challenge of collapsing the epilog for an irregular loop is two fold: recurrence constraints and register pressure (esp. prediate pressure). Recurrence

constraints (aka recurrence cycle, dependence cycle - should be defined in patent) refer to the order in which operations must execute. In processing the irregular loop so that the epilog can be collapsed, it is a challenge to minimize the number of new instructions added on the critical cycle. Lengthening this critical cycle slows down the loop by slowing down the rate at which new iterations can be initiated.

The contribution of applicant's invention is software pipelining irregular loops *without* the special purpose hardware and with a NON-rotating register file. This is achieved by the process claimed in claim 1 wherein it is determined which instructions in loop code in a memory may be speculatively executed, storing in a computer memory a set of registers that are modified by an instruction and are alive out of the loop, and modifying the program code so that the values of those registers are saved to a temporary register during all proper iterations, and copying back to a register the value of the temporary register once the loop is completed.

The minimum hardware that Rau_1 relies on is hardware for speculative execution. This is a non-significant, complex piece of hardware that is not realistic for all processors, especially many in the embedded world for a variety of reasons, including cost and power. These are not issues in the Rau world (general purpose processors). Their alternate hardware configuration has rotating register and hardware for speculative code motion. The method claimed is not taught in this reference.

Rau_2 depends on dynamic registers (aka rotating register files) or MVE (software-only approach) to software pipeline irregular loops. MVE has a big drawback: code size!! With MVE, loop unrolling (making multiple copies of a loop body and multiple epilogs are *required*, both *very* expensive in terms of code size.

Additionally, loop unrolling often increases register pressure significantly (from our experience - mileage may vary) which is problematic for loops that are already register-pressure bound. The method claimed is not taught by this reference

Akkary relies on trace buffers for speculative execution, again a non-trivial piece of hardware. Again, it's a different ballgame when you can put this type of hardware into your processor. The method is not taught by this reference.

The Bringmann thesis discusses compiler techniques implemented within the framework of the IMPACT compiler. Chapter 3 describes the concept of compiler-controlled speculation. Examples of speculation models are described for different speculation classes. Write-back suppression is presented in Chapter 4 to present an alternative compiler-controlled speculation model that requires some processor assistance to perform recovery. Chapter 5 describes an alternative speculation model that can be used to enhance the performance of existing speculation models. The examiner makes reference to page 52, chapter 4.3.2. on register allocation extension. The register allocator assumes that all locatable operands reside within virtual registers. For each of these virtual registers it constructs a live range which consists of the set of instructions where the operand is alive. Allocation then proceeds by coloring an interference graph constructed from these live ranges. It is not seen where this teaches applicants claimed method.

This is not taught or suggested by the references. The examiner argues the combination of Rau et al " Code Generation Schema for Modulo Scheduled Loops", ACM Proceedings of the 25th annual International Symposium on Microarchitecture,

Dec. 1992, hereinafter Rau_1 in view of Rau et al., "Register Allocation for Software Pipelines Loops" , June 1992, In Proc. of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2), and further in view of Akkary, USPN 6,240,509 (hereinafter Akkary) and Bringmann, "Enhancing Instruction Level Parallelism through compiler-controlled speculation", Univ. of Illinois, 1995 (hereinafter Bringmann). Nowhere in these reference is the method taught in these references. The examiner argues that in a method using speculation in handling pipelined loops is analogous to that of Rau_1, Bringmann discloses register extensions and Akkay discloses the use of temporary registers and instruction trace buffer for extension of the speculation- intensive pipelined in order to prevent mis-speculation recovery resources. The examiner then concludes that this teaches the committing or copying of values to and from secondary registers to provide for mishaps recovery and roll back situations from secondary registers to provide for mishaps recovery and rollback situations from the course of taking a wrong path during execution. The examiner argues that when register resources are available, it would have been obvious for one of ordinary skill in the art at the time the invention was made to implement to 2-copies techniques with hardware support by Rau_2) the use of register support such as register extension by Bringmann, or temporary registers as taught by Akkay, because this would allow extended means to provide for mishaps recovery and rollback situations so to make optimal use of architectural resources as intended by both Rau_1 and Rau_2. This whole argument is not based on the teaching of the references themselves but on an examiners hindsight reconstruction using references that do not point to such a combination. The explanation and the number of references involved

coupled by the examiners need to rely on his speculative combinations makes it clear that the subject invention is not obvious in view of the references. The method is clearly not taught in these references.

Furthermore, in regard to combining the cited prior art, reference is made to In re Fritch, 23 USPQ2d 1780 and particularly the portion thereof at page 1783 under "Prima Facie Obviousness" where the Court stated:

> "In proceedings before the Patent and Trademark Office, the Examiner bears the burden of establishing a prima facie case of obviousness based upon the prior art. '[The Examiner] can satisfy this burden only by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the references.' The patent applicant may then attack the Examiner's prima facie determination as improperly made out, or the applicant may present objective evidence tending to support a conclusion of nonobviousness.'"

and later stated:

> "'Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching or suggestion supporting the combination. Under section 103, teachings of references can be combined only if there is some suggestion or incentive to do so.' Although couched in terms of combining teachings found in the prior art, the same inquiry must be carried out in the context of a purported obvious 'modification' of the prior art. The mere fact that the prior art may be modified in the manner suggested by the Examiner does not

make the modification obvious unless the prior art suggested the desirability of the modification."

There is no suggestion in the references to suggest the method claimed or the desirability of the modification.

Claim 1 is therefore deemed allowable over the references.

Claims 2 and 3 dependent on claim 1 is deemed allowable for at least the same reasons as Claim 1.

Claim 4 calls for : A method for software pipelining of irregular conditional control loops includes pre-processing the loops so they can be safely software pipelined, comprising the steps of:

pre-processing each instruction in the loop in turn;

if the instruction can be safely speculatively executed, leaving the instruction alone;

if it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction using register copying and otherwise using predication.

The method of preprocessing is not taught or suggested in the references. The references don't apply for the same reasons as Claim 1 as to register copying. The process claimed teaches if the instruction can be safely speculatively executed, leaving the instruction alone and if it could be safely speculatively executed except that it modifies registers that are live out of the loop, preferably pre-processing the instruction using register copying and if that does not result in the best results using predication. There is no suggestion of this method in the references. The prior art techniques lead to drastic increases in code. There are a limited number of slots and we MUST avoid drastic

increases in code size. The prior art does not limit the code size like applicant's claimed method.

CONCLUSION

In view of the above, the examiners rejection of Claim 1 under 35 U.S.C. §112, first paragraph, as failing to comply with the written description requirement should be reversed.

In view of the above, the examiners rejection of Claim 1 under 35 U.S.C. §112, second paragraph, as failing to comply with the written description requirement should be reversed.

In view of the above, the examiner rejection of Claims 1-4 under 35 U.S.C. 103 (a) as being unpatentable over over Rau et al " Code Generation Schema for Modulo Scheduled Loops", ACM Proceedings of the 25th annual International Symposium on Microarchitecture, Dec. 1992, hereinafter Rau_1 in view of Rau et al., "Register Allocation for Software Pipelines Loops" , June 1992, In Proc. of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2), and further in view of Akkary, USPN 6,240,509 (hereinafter Akkary) and Bringmann, "Enhancing Instruction Level Parallelism through compiler-controlled speculation", Univ. of Illinois, 1995 (hereinafter Bringmann) should be reversed.

Respectfully submitted,
Robert L. Troike
Robert L. Troike

Reg. No. 24183
(972) 917-4360

APPENDIX

1. (currently amended): A method of pipelining program loops having irregular loop control comprises the steps of:

determining which instructions in loop code in a memory may be speculatively executed without special hardware support or special loop control instruction,

storing in a computer memory a set of registers that are modified by an instruction and are alive out of the loop, and

modifying the program code so that the values of those registers are saved to a temporary register during all proper iterations, and

copying back to a register the value of the temporary register once the loop is completed.

2. (original): The method of Claim 1, applied to any program loops wherein a minimum trip count is reduced to one.

3. (original): The method of Claim 1, applied to any program loops wherein a need for a multiversion code is eliminated.

4. (previously amended): A method for software pipelining of irregular conditional control loops includes pre-processing the loops so they can be safely software pipelined, comprising the steps of:

pre-processing each instruction in the loop in turn;

if the instruction can be safely speculatively executed, leaving the instruction alone;

if it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction using register copying and otherwise using predication.

5. (cancelled) The method of Claim 4, including the step of pre-processing the instruction by predication if it could not be safely speculatively executed.

6. (cancelled) The method of Claim 4, wherein if it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction by register copying.